

A Bayesian Approach to Predicting NFL Quarterback Scores in Fanduel Tournaments

STAT 578, Fall 2017, Team 5: Aaron Ray, Kiomars Nassiri, Michael Chan

October 25, 2017

Project Description

The National Football League (NFL), being one of the major professional sports leagues in North America, has a wide audience. participates in the NFL craze by competing in fantasy football tournaments organized by the daily fantasy site, “FanDuel.com”. Participants in a **Fantasy Football** game act as the managers of a virtual football team and try to maximize their points by picking up the best line-up. Points are given based on actual performance of players in real-world competition. For the purpose of this project we have chosen to work with the data gathered from the **FanDuel** internet company. We will leverage a Hierarchical Bayesian approach with the Markov Chain Monte Carlo method to predict the fantasy points likely to be scored by an NFL quarterback in any given game. The goal is to predict the points scored by each player given certain prior conditions and predictor variables that will assist our model in providing credible posterior prediction intervals.

The analysis is inspired by the study presented in the article, **Bayesian Hierarchical Modeling Applied to Fantasy Football Projections for Increased Insight and Confidence**, by Scott Rome.

Team Members

- **Aaron Ray** (aaronwr2@illinois.edu)*
- **Kiomars Nassiri** (nassiri2@illinois.edu)
- **Michael Chan** (mhchan3@illinois.edu)

*Contact Person

Dataset Description

Team has set up a process to gather the historical data from the RotoGuru website. The following is the code used to get the data from RotoGuru:

```
# Scrape rotoguru1 site for weekly FanDuel stats and bind each week's data to the
# pre-defined dataframe, 'd'.

for(year in 2014:2017){
  for(week in 1:16){
    page = read_html(
      gsub(" ", "", ,
           paste("http://rotoguru1.com/cgi-bin/fyday.pl?week=", week, "&year=",
                 year, "&game=fd&scsv=1"))
    )
    dtext = page %>% html_nodes("pre") %>% html_text(trim = TRUE)
    dtable = read.table(text=dtext, sep = ";", header=TRUE, col.names = cnames,
```

```

    quote=NULL)
d = rbind(d,dtable)
}
}

```

Data cleaning is performed using R routines. Some data cleaning tasks are needed to calculate Player rank.

Response Variables

- **FanDuelPts**: Points position at the end of a single game

Predictor Variables

- **6GmAvgOppPAP**: The six game average Opposing Points Allowed to Position (OppPAP) by the current player's opposing defense. For example, if the Buffalo Bills defense allowed a total of 30 points per game to wide receivers for six games straight, then this number would equal to the average of 30 for any wide receiver facing the Bills defense.
- **Position**: The position the player plays
- **Opponent**: The team that the player plays against
- **Rank**: The rank of a player based on recent performance

Analysis Ideas

Model

At the lowest level, we model the performance (**FanDuelPts**) as normally-distributed around a true value:

$$y | \alpha, \beta_{defense}, \beta_{home}, \beta_{away}, \sigma_r^2 \sim N(\alpha + X_{defense} \cdot \beta_{defense} + X_{home} \cdot \beta_{home} + X_{away} \cdot \beta_{away}, \sigma_y^2 I)$$

where

$$\alpha = \text{OffRnk5Wks}$$

$\beta_{defense,t,p}$ = defense coefficient against team t for position p

$\beta_{home,p,r}$ = home coefficient for position p and a rank r player

$\beta_{away,p,r}$ = Away coefficient for position p and a rank r player

$$y = \text{FanDuelPts}$$

$x_{t,p}$ = interaction indicator term for team t, position p

$x_{p,r}$ = interaction indicator term for rank r, position p

At higher level, we model the defense effect, $\beta_{defense}$, as how good a player is when playing against a particular team. We pool the effect based on the position of the player. That is, the defense coefficient is normally distributed from the same position specific distribution.

$$\beta_{defense,t,p} \sim N(\delta_p, \sigma_\delta^2)$$

For the home and away game effect, β_{home} and β_{away} , we model the effect for player of the same rank has the same distribution.

$$\beta_{home,r} \sim N(\eta_r, \sigma_\eta^2)$$

$$\beta_{away,r} \sim N(\rho_r, \sigma_\rho^2)$$

We will approximate non informative prior using:

$$\sigma_y \sim Inv - gamma(0.0001, 0.0001)$$

$$\sigma_\delta \sim N(0, 100^2)$$

$$\sigma_\eta \sim N(0, 100^2)$$

$$\sigma_\rho \sim N(0, 100^2)$$

Here is the JAGS model:

```

sink("fdp.bug")
cat("
model {
  for (i in 1:length(y)) {
    y[i] ~ dnorm(alpha[i] + inprod(X.defense[i, ], beta.defense)
                  + inprod(X.home[i, ], beta.home)
                  + inprod(X.away[i, ], beta.away), sigmasqinv)
  }

  # The entry of the beta.defense corresponds to Opponent:Position
  # In our model, we pool the beta.defense based on position.
  # i.e. All defense effects of the same position are drawn from the same distribution
  for (p in 1:Num.Position) {
    for (t in 1:Num.Opponent) {
      beta.defense[(p-1) * Num.Opponent + t] ~ dnorm(delta[p], 1/100^2)
    }
    delta[p] ~ dnorm(0, 1/100^2)
  }

  # The entry of the beta.home and beta.away corresponds to Rank:Position
  # In our model, we pool the beta.home/away based on rank
  for (r in 1:Num.Rank) {
    for (t in 1:Num.Position) {
      beta.home[(r-1) * Num.Position + t] ~ dnorm(eta[r], 1/100^2)
      beta.away[(r-1) * Num.Position + t] ~ dnorm(rho[r], 1/100^2)
    }
    eta[r] ~ dnorm(0, 1/100^2)
    rho[r] ~ dnorm(0, 1/100^2)
  }

  sigmasqinv ~ dgamma(0.0001, 0.0001)
  sigmasq <- 1/sigmasqinv
}

",fill = TRUE)

## 
## model {
##   for (i in 1:length(y)) {
##     y[i] ~ dnorm(alpha[i] + inprod(X.defense[i, ], beta.defense)
##                   + inprod(X.home[i, ], beta.home)
##                   + inprod(X.away[i, ], beta.away), sigmasqinv)
##   }
## 
##   # The entry of the beta.defense corresponds to Opponent:Position

```

```

##  # In our model, we pool the beta.defense based on position.
##  # i.e. All defense effects of the same position are drawn from the same distribution
##  for (p in 1:Num.Position) {
##    for (t in 1:Num.Opponent) {
##      beta.defense[(p-1) * Num.Opponent + t] ~ dnorm(delta[p], 1/100^2)
##    }
##    delta[p] ~ dnorm(0, 1/100^2)
##  }
##
##  # The entry of the beta.home and beta.away corresponds to Rank:Position
##  # In our model, we pool the beta.home/away based on rank
##  for (r in 1:Num.Rank) {
##    for (t in 1:Num.Position) {
##      beta.home[(r-1) * Num.Position + t] ~ dnorm(eta[r], 1/100^2)
##      beta.away[(r-1) * Num.Position + t] ~ dnorm(rho[r], 1/100^2)
##    }
##    eta[r] ~ dnorm(0, 1/100^2)
##    rho[r] ~ dnorm(0, 1/100^2)
##  }
##
##  sigmasqinv ~ dgamma(0.0001, 0.0001)
##  sigmasq <- 1/sigmasqinv
## }
##

```

```
sink()
```

Sample Data

```

fdp <- read.csv("fdpfinal.csv", sep = ',', header = TRUE)

head(fdp)

##   Position Year YearWeek Opponent Week PlayerId          Name Team
## 1       QB 2015     201513  Steelers  13    1060 Hasselbeck, Matt  Colts
## 2       QB 2015     201514  Jaguars  14    1060 Hasselbeck, Matt  Colts
## 3       QB 2015     201515   Texans  15    1060 Hasselbeck, Matt  Colts
## 4       QB 2015     201516  Dolphins 16    1060 Hasselbeck, Matt  Colts
## 5       QB 2015     201501   Ravens  1    1081 Manning, Peyton Broncos
## 6       QB 2015     201502   Chiefs  2    1081 Manning, Peyton Broncos
##   HomeGame FanDuelPts FanDuelSalary AvgOppPAP7Wks SdOppPAP7Wks OallAvgPAP
## 1         0        6.86        6500      20.95      8.045    17.44
## 2         0        9.08        6600      24.16      6.738    17.44
## 3         1        8.98        6400      16.36      9.690    17.44
## 4         0        3.96        6000      20.46      6.002    17.44
## 5         1        5.90        9100      18.99     11.697    17.44
## 6         0       21.24        8200      13.85      4.342    17.44
##   OallStdevPAP AvgPts5Wks StdevPts5Wks OffRnk5Wks DefRnk7Wks
## 1      2.909     14.85      4.824      Rank1      Rank2
## 2      2.909     14.82      4.897      Rank1      Rank2
## 3      2.909     13.56      5.490      Rank1      Rank3

```

```

## 4      2.909      12.11      5.566    Rank1    Rank2
## 5      2.909      14.96      8.496    Rank1    Rank2
## 6      2.909      10.53      5.011    Rank2    Rank3

```

```

fdp['Rank'] = fdp$OffRnk5Wks
fdp['Locality'] = 'Away'
fdp[fdp$Home.Game == 1, 'Locality'] = 'Home'

```

** The following R code is for reference and preliminary research **

```

fdp_train=fdp[fdp$Year == 2015, ]

X.defense = model.matrix(~ 0 + Opponent:Position , data=fdp_train)
X.home = model.matrix(~ 0 + Rank:Position , data=fdp_train)
X.away = model.matrix(~ 0 + Rank:Position , data=fdp_train)

X = cbind(X.defense, X.home, X.away)

Num.Opponent = length(unique(fdp[, "Opponent"]))
Num.Position = length(unique(fdp[, "Position"]))
Num.Rank = length(unique(fdp[, "Rank"]))

```

Simple Ideas

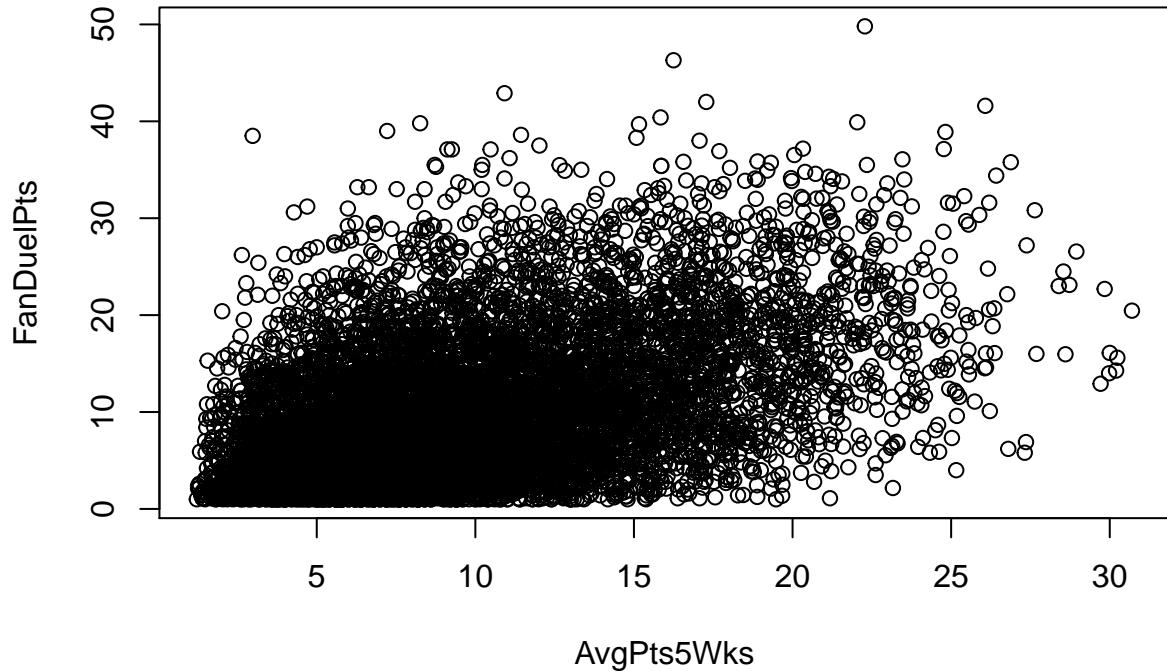
$$y|\alpha \sim N(\alpha, \sigma_y^2 I)$$

```

mod.classic = lm(FanDuelPts ~ AvgPts5Wks, data = fdp)

plot(FanDuelPts ~ AvgPts5Wks, data = fdp)

```



All over the place, let's add the team defense

X.defense is the indicator matrix

```

library(rjags)
set.seed(20171008)

# Initialization List for the 4 chains
jags.inits=list(
  list( sigmasqinv=    0.01,
        .RNG.name = "base::Mersenne-Twister", .RNG.seed = 20171008 ),
  list( sigmasqinv=    0.01,
        .RNG.name = "base::Mersenne-Twister", .RNG.seed = 20171008 + 1 ),
  list( sigmasqinv=0.000001,
        .RNG.name = "base::Mersenne-Twister", .RNG.seed = 20171008 + 2 ),
  list( sigmasqinv=0.000001,
        .RNG.name = "base::Mersenne-Twister", .RNG.seed = 20171008 + 3 )
)

data.jags <- list(
  y= fdp_train$FanDuelPts,
  alpha = fdp_train$OffRnk5Wks,
  X.defense = X.defense,
  X.home = X.home,
  X.away = X.away,
  Num.Position=Num.Position,
  Num.Opponent=Num.Opponent,
)

```

```

    Num.Rank=Num.Rank
}

runModel=FALSE

if (runModel) {
  m <- jags.model("fdp.bug", data.jags, inits = jags.inits, n.chains=4, n.adapt = 1000)
  update(m, 2500) # burn-in

  x <- coda.samples(m, c("delta", "eta", "rho", "beta.defense", "beta.home", "beta.away", "sigmasq"), n
  save(file="fdp.jags.model.Rdata", list="m")
  save(file="fdp.jags.samples.Rdata", list="x")
} else {
  load("fdp.jags.samples.Rdata")
}

```

Check convergence

```
plot(x, smooth=FALSE)
```

